Associations in NIEM 0.2

Georgia Tech Research Institute

December 20, 2005

Contents

1 Requirements

1. Represent associations among objects such that they more closely follow the relational data model.

2. Represent associations as reusable types in XML Schema
3. Represent properties of an association, which are not properties of the participants in an association
4. Represent n-ary associations when appropriate.
5. Identify the types of the base objects in an association.

2 Introduction

The GJXDM 3.0 provided many data definitions. These consisted primarily of types and properties. Types were used to represent real-world objects. Properties were used to represent connections between these objects. These connections included:

Characteristics: Values that are specific to an object, and likely invariants of that object

Subparts: Objects that are smaller pieces of other objects

Relationships: Connections between objects, which may be numerous and changing

GJXDM 3.0 represented all of these connections between objects in the same form, as RDF-like object-property-object triplets. These triplets were represented by two main XML forms: content elements and reference elements. Both of these forms were available for each property of an object.

This proposal provides a recasting of the way types are used, and provides for narrowing of the XML syntax for representing properties in the data model. This should result in an easier to understand XML syntax. It should enhance understanding by users familiar with relational databases, as it provides a representation similar to relational databases, while maintaining the power of the more flexible GJXDM 3.0.

Under this proposal, types will not be used only for representation of objects. In addition to object definitions, types will be defined to represent the association between objects. These are called association objects, and the types are association types. Associations will take the place of many GJXDM 3.0 properties.

Also outlined by this proposal is a process for simplifying the representation of a property of an object. Where all properties were available under GJXDM 3.0 as either reference or content elements, under this proposal only one form will be selected for each property, when possible. Some properties will be selected to be content elements. Other properties will be defined as reference elements. Only special exceptions will maintain both forms, on an as-needed basis.

By making these changes, the data model will achieve a more relational basis. Under the relational model, tables may be used to define objects, or to

define join tables linking multiple objects. Associations fill the role of such join tables. Similarly, in the relational model, a table either contains a value, or contains a foreign key that references a value in another table. These correspond to the content elements and reference elements of this proposal. This correspondence should lead to a smooth normalization process, and greater community acceptance and understanding of the data model.

3 Description of Technique

This technique provides simplifications and additional features to the GJXDM and NIEM. Specifically, the techniques involve:

1. Addition of constructs to represent associations between objects
2. Normalization of the model to incorporate associations
3. Normalization of the model to select either reference or data elements, when possible, instead of the GJXDM option of making both available.

3.1 Association Instance Syntax

The syntax for an instance of an association is simple. Take, for example, the marriage of Adam and Barbara Smith:

```
<MarriageAssociation>
  <SpouseRef s:ref="A"/>
  <SpouseRef s:ref="B"/>
  <MarriageDate>1937-05-12</MarriageDate>
  <DivorceDate>1973-06-02</DivorceDate>
</MarriageAssociation>
```

Interpreting the above XML fragment is straightforward:

- There is an association that we call a marriage. You can tell it is an association, and not a thing, because it is named "*something association*".
- This marriage association has two spouses, a marriage date, and a divorce date.
- One spouse is referenced as the object with the identifier `A`. The other spouse is identified by the ID `B`.

These objects are specified elsewhere in the same XML instance: Object A is specified as follows:

```
<Person s:id="A">
  <PersonName>
    <PersonFullName>Adam Smith</PersonFullName>
  </PersonName>
</Person>
```

Object B is specified as follows:

```
<Person s:id="B">
  <PersonName>
    <PersonFullName>Barbara Smith</PersonFullName>
  </PersonName>
</Person>
```

Other elements in the association specify more information about the association:

```
  <MarriageDate>1937-05-12</MarriageDate>
  <DivorceDate>1973-06-02</DivorceDate>
```

The marriage date and divorce date are specific to the relationship between the two spouses, and so is a natural fit for an element of the association.

3.2 Multiple Associations

An object may be involved in multiple associations, each of which is represented independently. The examples below all occur within a single XML instance, and all refer to the same object with identifier `A`. In this case, the object `A` is a person, who is an employee, a spouse, a parent, and a child.

```
<EmployerEmployeeAssociation>
  <EmployeeRef s:id="A"/>
  ...
</EmployerEmployeeAssociation>

<MarriageAssociation>
  <SpouseRef s:id="A"/>
  ...
</MarriageAssociation>

<ParentChildAssociation>
  <ParentRef s:id="A"/>
  ...
</ParentChildAssociation>

<ParentChildAssociation>
  <ChildRef s:id="A"/>
  ...
</ParentChildAssociation>
```

3.3 Schema for Associations

The definition of an association is composed of several parts:

1. An element that identifies the specific semantics of the association.
2. A type for the association. The type may be have precise semantics, or may be a more generally defined type.

3.3.1 Element definitions

For each semantically distinct association, we define an element. Each element will have annotations indicating the specific meaning of the association. Such documentation is not shown in this document, but follows the guidelines established for GJXDM 3.0. The syntax is standard XML Schema. For example, here is the definition for a parent-child element:

```
<xs:element
    name="ParentChildAssociation"
    type="ParentChildAssociationType"/>
```

We may wish to define a more-specific type of parent-child association. For example, an adoptive parent-child association:

```
<xs:element
    name="AdoptiveParentChildAssociation"
    type="ParentChildAssociationType"/>
```

If we wanted to make the type specific to an adoptive parent-child situation, then we define a new type, instead of reusing the general parent-child type.

3.3.2 Association type definitions

The definition of types for associations is done as needed, depending on the content of the types. We do not, as a rule, define a new type for each use or semantic definition of an association. Instead, we define them as necessary, to accommodate the content required. Here is an example definition for a type for the parent-child association:

```
<xs:complexType name="ParentChildAssociationType">
  <xs:complexContent>
    <xs:extension base="u:AssociationType">
      <xs:sequence>
        <xs:element ref="this:ParentRef" minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element ref="this:ChildRef" minOccurs="0"
            maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The type definition has several parts:

1. The name of the type is "*something AssociationType*". This makes associations between objects distinct from other types of object definitions.
2. The type is derived from another association type. This allows definition of type hierarchies for associations, and the definition of characteristics that are shared across multiple association types.
3. The content of the association is a sequence of elements. The content of the association could be entirely related objects. The association could also contain characteristics of the associations, such as dates, names, identifiers, etc.

3.4 Association type hierarchy

The use of a type hierarchy is a useful feature, but should not be overused. In the examples so far, we have seen the following:

1. A root *association type*, which helps group association types.
2. An association type for a parent-child association. This type had a parent and a child.
3. An association type for the marriage association.

We may wish to insert into this list of types a root type for all interpersonal associations. This, however, may be over-design, due to several factors:

1. What content would go into a generalized interpersonal association? All we would know is that the participants were people. A list of `PersonRef` elements is not very useful, and does not provide any semantics. The elements defined at this stage would have to be discarded to provide concrete meaning (such as spouse, parent, and child).
2. What makes an association interpersonal? Is it just that there are two people participating in the association? Would an employer-employee be an interpersonal relationship, if the employer were an individual? Would an offender-victim relationship be interpersonal? What if the victim was an organization?
3. Due to restrictions of XML Schema, we only have single-inheritance available in our toolbox; a type may have at most only a single parent. These sorts of *place-holder* types have limited usefulness, as they cannot be combined together to provide useful meaning.

Use of type inheritance should be carefully considered. Keep in mind that common types may be inserted into the type hierarchy later in model development.

4 Model Normalization

Many users found the GJXDM 3.0 schemas difficult to use, as they were not very constraining on the structure of XML that conformed to the model. Links between objects were kept as flexible as possible, to accommodate a wide variety of usage. Under NIEM 0.2, the data model will be simplified, to take advantage of users' familiarity with the relational model, and to simplify usage patterns.

This normalization takes two forms:

1. Separating associations between objects from the definitions of the objects themselves
2. Selection of either reference or local-content elements, when possible

Both normalization steps help the data model achieve a more regular structure. This should be more easily understood by people with experience with relational databases, and should provide for more uniform implementations.

4.1 Adapting Associations

There are numerous considerations in adapting and defining associations. We need a method of distinguishing associations from other methods of connecting objects. We must understand to what degree we should specialize or generalize associations. Third, we need procedures for refactoring the GJXDM 3.0 data model to accommodate associations.

4.1.1 The Litmus Test

A type is an association among objects (i.e. an AssociationType should be created to relate the objects) if and only if:

1. The related objects are peers of one another and not simply a defining characteristic of or subpart of the other object(s). The term peers is used in a data modeling sense to mean that each object being related has its own set of characteristic property values independently of the other.
2. Each related object can exist independently, that is, it does not dependent on the existence of the association or the other object(s). In other words, none of the objects being related should lose meaning if separated from the others.
3. The association has its own characteristic attributes (properties) that either cause or result from the existence of the association. These attributes are characteristic of the association and define its nature or distinguish it from other associations and objects.

4.1.2 Guidelines

1. New associations should be identified based on requirements or use within IEPDs, not simply because they exist, or may be used someday.
2. Exploit the relational properties that are already in the model (particularly within the PersonType and ActivityType)
3. To avoid enumerating all associations, there may be general association types to create as the basis for others (e.g., Association, Kinship, etc.)
4. Conversion of each simple pair-wise property to an association will require consensus on names for:
   (a) The association itself
   (b) All objects related by the association
   (c) The properties of the association

4.2 Procedure for adopting associations

There are several steps involved in adopting associations. These are:

1. Remove the GJXDM 3.0 RelationshipType. This has the side effect of eliminating the terms subject and object from the schemas.
2. Create a new AssociationType to act as the base type for all relationships.
3. For each existing type in the current model:
   (a) Decide if the type represents a relationship (association) or an object by applying the litmus test.
   (b) If object, leave it to be derived from SuperType or a type derived from SuperType.
   (c) If association, derive it instead from AssociationType, or a type derived from AssociationType.

5 Normalizing Element Uses

The next step in normalizing the data model is selecting the representation for specific element uses. Under GJXDM 3.0, all elements within types were represented two ways:

1. A content element
2. A reference element.

5.1 Content Elements

Content elements enclose data. The following is an example:

```
<Person s:id="A">
  ...
  <PersonName>
    <PersonFullName>Adam Smith</PersonFullName>
  </PersonName>
  ...
</Person>
```

In this example, there is a person object. The person contains an element called `PersonName`. The PersonName element contains an element called `PersonFullName`. The PersonFullName element contains a string `Adam Smith`. The PersonFullName element is obviously a content-containing element. It has the person's name (a literal string) as its content.

The PersonName is also a content-containing element, as its content represents the person name, as a structured object. It contains the element PersonFullName, and could contain additional elements.

5.2 Reference Elements

Reference elements do not enclose content. Instead, they reference content as external objects:

```
<Incident>
  <ActivityDate>2003-10-02</ActivityDate>
  ...
  <IncidentSeizedPropertyRef s:ref="C"/>
  ...
</Incident>
```

In the above example, the property that was seized as part of the incident is referenced out to another object, an XML object in the same XML instance, with the identifier `C`.

```
<Property s:id="C">
  <PropertyDescriptionText>
    White microwave oven
  </PropertyDescriptionText>
  <PropertyTypeCode>HOVEN</PropertyTypeCode>
  <PropertyMakeName>Kenmore</PropertyMakeName>
  <PropertyModelName>63292</PropertyModelName>
</Property>
```

The object that has the identifier `C` is an instance of `Property`, specifically representing a microwave oven. The reasons for representing the microwave oven

outside of the incident should be quite evident: it is its own object, independent of the incident. It has its own life cycle. If the incident did not exist, the microwave oven would still exist.

The seized property is an element of the incident because it is a fixed part of the incident. The incident involved the seizing of the property, and that will not change. However, the incident should be a reference element, as the property has its own life cycle, outside of the incident.

5.2.1 Identifying types for reference elements

All reference elements are of the same XML Schema type: `ReferenceType` from the structures namespace. However, we would like to validate the XML Schema type of the thing to which the reference is referring (the *referred* object). For example:

```
<IncidentSeizedPropertyRef s:ref="C"/>
```

For `IncidentSeizedProperty`, we would like the XML Schema type of the referred object to be `PropertyType`, or something derived from that type. XML Schema does not help us here, because it does not support type checking of reference targets. XML Schema supports XML:ID and XML:IDREF types, but the constraints applied to them are few: no ID may be defined more than once, and any IDREF must refer to a defined ID. Beyond that, XML Schema does not help.

To define the type of referred objects, we add additional non-XSD information to the schema, which we may interpret with programs, stylesheets, or constraint languages. This additional information is added to the element definitions, and concretely specifies the type of referred objects.

```
<xs:element name="IncidentSeizedPropertyRef"
    type="s:ReferenceType">
  <xs:annotation><xs:appinfo>
    <i:referenceTarget i:name="PropertyType"/>
  </xs:appinfo></xs:annotation>
</xs:element>
```

In this example, the incident seized property is specifically defined to be of type `PropertyType` in the same namespace. Following XML Schema rules, we would expect the target of the reference to be of type PropertyType, or of a type properly derived from PropertyType.

5.3 Procedure for Normalizing Elements

For each existing element occurring in a type:

1. If the element links to a peer object, or to an independent object, then define it as a reference element

2. If the element constitutes a characteristic or subpart of the containing object, then define it as an in-line content element
3. If the element should be an association, then
   (a) remove it from the containing type
   (b) create a new association type for it
   (c) add the containing type as a related object
   (d) add the type of the original element as a related object, and
   (e) add properties for the association, as needed

6 Advantages

1. Relationships modeled as associations are similar to the relational data model.
2. All relationships have their own type definitions.
3. Many objects become smaller and have simpler content to represent.
4. There are fewer alternative representations. Each component is fixed as either in-line or referenced content. (Of course, this also requires a decision for each component).
5. Adding an association has minimal or no impact on other objects.
6. The data model might be more directly represent-able in UML (because it is more relational).

7 Disadvantages

1. References to IDs are more difficult to process, and will add overhead
2. IDs are required for all reference-able components.
3. XML Schema cannot validate that the references to object IDs are valid objects with the correct types.
   (a) The structure of both objects and associations can still be validated independently.
   (b) Other means besides XML Schema could be used to validate the types of referenced objects.